2003

Learning Monotone DNF Using SPP Teaching Assistant

N. V. Vinodchandran
University of Nebraska-Lincoln, vinod@cse.unl.edu

University of Nebraska-Lincoln, Computer Science and Engineering
Technical Report # TR-UNL-CSE-2003-0008

Learning Monotone DNF Using SPP Teaching Assistant

N. V. Vinodchandran

University of Nebraska-Lincoln
Lincoln, NE 68588, USA
email: vinod@cse.unl.edu

Abstract

In this paper we prove a new upper bound on learning monotone DNF formulae in an exact learning model called the *teaching assistant model*. This model, introduced in one of our earlier works, uses teaching assistant classes for classifying the complexity of learning problems. Teaching assistant classes are a learning-theoretic analog of complexity classes. We show that monotone DNF are learnable using a teaching assistant in the class SPP. On the other hand, concept classes such as k-CNF or Boolean circuits are not learnable using an SPP teaching assistant unless NP $\subseteq$ SPP. We also observe that the recent SPP algorithm for Graph Isomorphism problem due to Arvind and Kurur can be adapted to learn the concept class of permutation groups using an SPP teaching assistant. This solves an open problem from our earlier work.

Keywords: Exact learning, Complexity of learning.

1 Introduction

This paper deals with the complexity of learning. Studying the inherent complexity of learning problems is a major concern in computational learning theory. Angliun's exact learning model [Ang88] is a suitable model for such a study. In this model a learner learns a concept by querying an honest teacher about the target concept. Two types of queries that are well studied are equivalence queries and membership queries. In this framework, one way to quantify the complexity of a concept class is to consider the type and number of queries that a learner has to ask the teacher to learn any concept in the concept class. This is known as the query complexity. Hence, for example, if any learner requires many more equivalence queries to learn the concept class $\mathcal{C}_1$ than to learn $\mathcal{C}_2$, we can say that $\mathcal{C}_1$ is more difficult to learn than $\mathcal{C}_2$. Studying the query complexity in Angluin's model continues to be a very active and fruitful research area ([Ang90, Gav93, HPRW96, Heg95, BCG00, BCG01]).

There are limitations to this query-based approach in studying the complexity of learning problems. For example, how do we compare the query complexity of two concept classes that are both learnable with polynomially many equivalence queries? It could be the case that one of these concept classes is "easier" than the other because the full power of equivalence queries may not be needed to learn one, but is required to learn the other. For proving finer classifications among learning problems with respect to their complexity, a refinement of

Angluin's exact learning model called the *teaching assistant* model was developed in [AV96, AV00, Vin99]. This model is based on the complexity-theoretic notion of *complexity classes*. Before we discuss the new results in this paper, we informally explain the teaching assistant model and our prior research in this model.

The Teaching Assistant model: A model based on complexity classes

A central idea in this model is the notion of a *teaching assistant*; an intermediate agent between the learner and the teacher. The learner, instead of communicating directly with the teacher, communicates with a teaching assistant to learn the target concept. The teaching assistant in turn communicates with the teacher to retrieve information about the target concept. The power of this model for investigating the complexity of learning problems comes from the fact that we can classify teaching assistants into *teaching assistant classes*, which is similar to the way we classify computational problems into complexity classes.

The development of the idea of teaching assistants was based on an observation connecting the complexity class NP with equivalence queries in Angluin's model. An equivalence query to a teacher can be replaced by a sequence of queries to an NP oracle (essentially for prefix-searching for a counter example), which in turn makes membership queries to the teacher. We can think of this NP oracle as an intermediate agent between the learner and the teacher. This NP oracle is an example of a teaching assistant. Hence, we can say that any concept classes learnable with equivalence queries can also be learned using an NP teaching assistant. Similarly, a membership query by the learner can be seen as a query to a teaching assistant in the class P. Therefore, any concept class that is membership query learnable is also learnable with a P teaching assistant.

The main advantage of restating the query learnability in the language of NP and P is that, now we can generalize this idea to other complexity classes. For example, consider Valiant's complexity class UP [Val76]. UP is the class of decision problems accepted by nondeterministic machines with a restriction that there is at most one accepting computation for any instance of the problem. This important complexity class has connections with cryptography [GS88]. The complexity of problems in UP is believed to be strictly between P and NP. That is, we believe that $\mathrm{P} \subsetneq \mathrm{UP} \subsetneq \mathrm{NP}$. Hence we can say that concept classes that are learnable using a teaching assistant in the class UP is less complex than concept classes that require assistants from the class NP. Thus, in a certain well-defined sense, the complexity of teaching assistants quantify the complexity of learning. In general, we can define a *teaching assistant class* analogous to any complexity class. Hence, introducing this intermediary called the teaching assistant in the exact learning model provides a framework which is suitable for analyzing the complexity of learning problems. We call this framework the teaching assistant model (in short TA model) for exact learning. A schematic diagram of the TA model in comparison with Angluin's exact learning model is shown in Figure 1.

In this paper we are interested in the teaching assistant class analogous to a complexity class called SPP. Next we briefly discuss the complexity class SPP and its significance in complexity theory.

The complexity class SPP and its significance

SPP was introduced and studied by Fenner, Fortnow, and Kurtz in the paper [FFK94]. (This class is also independently studied in [OH93] under the name XP and in [Gup95] under

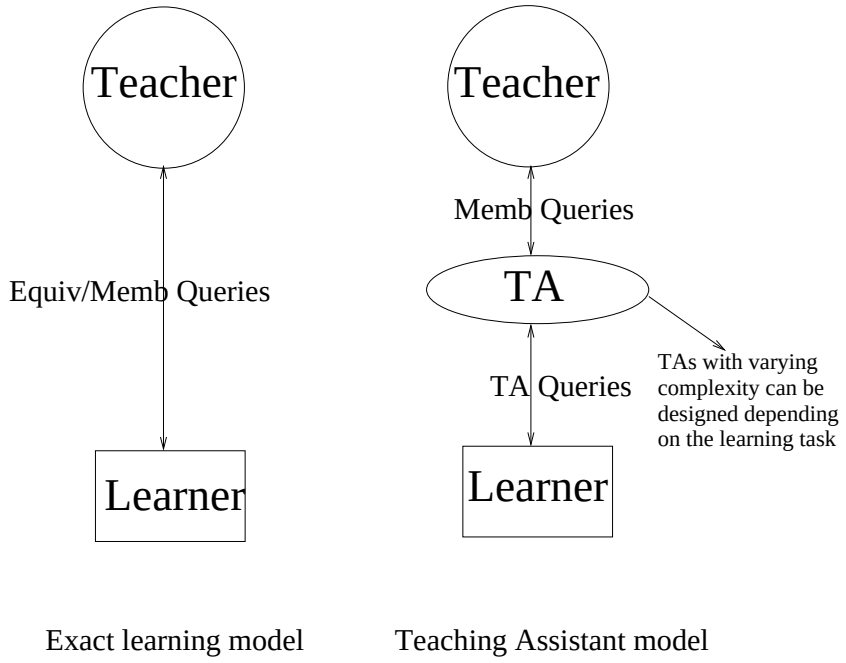

Figure 1: The teaching assistant model in comparison with the exact learning model

the name $\mathcal{Z}$UP). SPP consists of decision problems that are accepted by nondeterministic polynomial time Turing machines such that; for the positive instances, the difference in the number of accepting and rejecting paths of the machine is exactly one, and for the negative instances, this difference is zero. SPP can be seen as the *gap* (difference between accepting and rejecting paths) analog of the class UP.

From a structural point of view, this class is interesting mainly because of its *lowness* properties. A complexity class C_1 is said to be *low* for another (relativizable) complexity class C_2 if $C_2^{C_1} = C_2$. That is, any C_2-machine that uses a language in C_1 as oracle can be converted into another C_2-machine with the same acceptance behavior without any oracle access: C_1 is powerless as oracle to C_2. It is shown in [FFK94] that SPP is low for counting complexity class such as PP, $\mathrm{C_{=}P}$ and $\mathrm{Mod}_k\mathrm{P}$ for $k \geq 2$. Thus, intuitively, SPP is a class of "low counting complexity".

Although SPP is a structurally defined class, it has been proved to be very useful in analyzing the complexity of "intermediate problems": problems that are not known to be in P but unlikely to be NP-complete. Membership of a problem in the class SPP can be seen as an evidence that the problem is unlikely to be NP-complete: no known NP-complete problem is shown to be in SPP. In [KST92, Vin97, Vin99] it is shown that many of the known candidate problems of intermediate complexity from the domain of algebra are in the class SPP. Very recently, Arvind and Kurur [AK02] presented an elegant SPP algorithm for the well-known Graph Isomorphism problem.

The exact power of SPP in relation to NP is not yet clear. Because of the strong restrictions on the computation tree of nondeterministic Turing machines accepting SPP languages, it seems surprising that NP can be in SPP. Furthermore, there are oracles separating NP from SPP [Bei94]. On the other hand results from derandomization and resource bounded measure

theory give some evidence towards the containment of NP in SPP [KvM02, Hit03].

Prior results on teaching assistant model

In [AV96], the teaching assistant framework was introduced primarily to distinguish the complexity of learning certain algebraic concepts such as vector spaces and permutation groups from the "nonalgebraic" concept class 3-CNF. In particular, it is shown that vector spaces can be learned using teaching assistants in the class SPP. Learning 3-CNF using a teaching assistant in the class SPP is not possible, unless an unlikely collapse NP $\subseteq$ SPP happens. It is also shown that permutation groups can be learned using an assistant in the class LWPP: a generalization of the class SPP.

A subsequent work [AV00, Vin99] initiated a study of teaching assistant classes analogous to many well-studied complexity classes such as P, UP, NP, (UP $\cap$ co-UP), (NP $\cap$ co-NP), Σ_2^p, SPP, LWPP, and established their importance in classifying the complexity of learning. In particular, [AV00] gave a fine separation among many of the teaching assistant classes that are mentioned above, using sub concept classes of permutation groups. For example, [AV00] presented a concept class that is learnable using a (UP$\cap$coUP) teaching assistant, but not learnable using teaching assistants in P. Note that unlike in complexity theory where establishing proper separations among complexity classes is extremely difficult, we can actually show separations among TA classes.

New results in this paper and their significance

In this paper, we prove new upper bounds on the complexity of learning two concept classes; monotone DNF and permutation groups, in the TA model. We show that both these concept classes are learnable using teaching assistants in the class SPP.

These upper bound results are significant for several reasons. It follows from an earlier work [AV96] that there are concept classes such as k-CNF and Boolean Circuits, that are not learnable with an SPP assistant unless NP $\subseteq$ SPP (in the complexity-theoretic world). This establishes a separation between learnability of monotone DNF and k-CNF formulae (or Boolean circuits). Further, the upper bound for monotone DNF suggests that the TA model may be successfully employed to investigate the learnability of important and well-studied concept classes: our prior research in the TA model were focused mainly on concept classes that have some algebraic properties.

In complexity theory, the class SPP has been used successfully in studying computational problems with intermediate complexity. The results in this paper help to further establish the role of the class SPP in understanding the complexity of learning problems. Finally, the result on learning permutation groups solves an open problem stated in [AV00].

2 Notations, definitions, and technical preliminaries

In this section, we present relevant definitions related to complexity theory and learning theory. We begin with basic notations that we use throughout the paper.

2.1 Notations and basic definitions

We fix the finite alphabet $\Sigma = \{0,1\}$. $\langle .,. \rangle$ denotes any standard pairing function. Let $|w|$ denotes the length of a string $w \in \Sigma^*$. For sets X_1 and X_2, $X_1 \setminus X_2$ denotes their set difference. Let λ denotes the empty string. For $y_1, y_2 \in \Sigma^*$, $y_1 \leq y_2$ means that y_1 is lexicographically smaller than y_2. Let $\mathbf{Z}$ denotes the set of integers and $\mathbf{N}$ the set of natural numbers.

Complexity theory.

For any class of languages $\mathcal{L}$, co$\mathcal{L}$ denotes the class of languages L such that $\Sigma^* \setminus L$ is in $\mathcal{L}$. For a nondeterministic polynomial time (in short, NP) machine M, $acc_M : \Sigma^* \to \mathbf{N}$ denotes the function defined as: $acc_M(x)$ = the number of accepting paths of M on input x. #P denotes the class of functions $\{acc_M | M \text{ is an NP machine}\}$.

For an NP machine M, $gap_M : \Sigma^* \to \mathbf{Z}$ denotes the function defined as: $gap_M(x)$= number of accepting paths of M on input x – the number of rejecting paths of M on input x. GapP [FFK94] denotes the class of functions $\{gap_M | M \text{ is an NP machine}\}$. GapP is the closure of #P under subtraction. We will make implicit use of the following closure properties of GapP: a) product of a polynomial number of GapP functions is a GapP function, b) sum of a polynomial number of GapP functions is a GapP function (see [FFK94] for details).

The counting classes of interest in this paper is SPP. The class SPP consists of all those languages whose characteristic functions are in GapP. That is, a language L is in SPP if there is a function f in GapP such that; $f(x) = 1$ if $x \in L$ and $f(x) = 0$ if $x \notin L$. For the purpose of comparison, we give the definition of the class UP. A language L is in UP, if there is a function f in #P such that; $f(x) = 1$ if $x \in L$ and $f(x) = 0$ if $x \notin L$. Thus, SPP generalized the class UP. The inclusions P $\subseteq$ UP $\subseteq$ SPP are a direct consequence of the definitions. Other definitions from complexity theory used in the paper can be found in [BDG88].

Learning theory.

We only deal with the learnability of finite concepts. A *representation class* $\mathcal{P}$ is a tuple $\langle R, \mu, p \rangle$ where p is a polynomial, $R \subseteq \Sigma^*$ and μ is a collection $\mu = \{\mu_n\}_{n \in \mathbf{N}}$ such that $\mu_n : R \to 2^{\Sigma^{p(n)}}$ is a many-one mapping. Any element in the range of μ is called a *concept* of $\mathcal{P}$. The set of all concepts of $\mathcal{P}$ is called the *concept class* represented by $\mathcal{P}$. When there is no confusion, we denote the concept class represented by $\mathcal{P}$ also by $\mathcal{P}$. For any n, let $\mathcal{P}_n$ denote the concepts of $\mathcal{P}$ in $\Sigma^{p(n)}$. For any concept $c \in \mathcal{P}_n$ let $size(c)$ denote $\min\{|r| : \mu_n(r) = c\}$.

Before we define the teaching assistant model, we describe Angluin's exact-learning model [Ang88]. Let $\mathcal{P}$ be a representation class. In Angluin's model there is a deterministic learner which can query a teacher who selects a *target* concept $c \in \mathcal{P}_n$ and answers the query according to it. More specifically, the learner α on input 0^n, outputs a representation $r \in R$ such that $\mu_n(r) = c$, where c is the target concept. In the course of learning, α can make *equivalence queries* and *membership queries* to the teacher. For an equivalence query $h \in R$ made by α the teacher responds with 'Yes' if $\mu_n(h) = c$, or gives a *counterexample* $x \in \mu_n(h) \triangle c$ if $\mu_n(h) \neq c$. For a membership query $x \in \Sigma^{p(n)}$ made byα the teacher responds with 'Yes' if $x \in c$ and 'No' if $x \notin c$.

In this paper we consider *bounded learnability*. In bounded learnability, the learner is given, as part of the input, an upper bound on the size of the target concept. More precisely, for a representation class $\mathcal{P}$, the teacher selects a target concept $c \in \mathcal{P}_n$ and the learner α

takes as input a pair $\langle 0^n, 0^l \rangle$, and if $size(c) \leq l$, the learner has to output a representation $r \in R$ such that $\mu_n(r) = c$. The learner's output is unspecified if $size(c) > l$.

We now look at some standard complexity measures on the resources of the learner. Let $\mathcal{P}$ be a representation class. Let α be a learner learning $\mathcal{P}$. For an input 0^n, the *query complexity* of a learner is the sum of the lengths of equivalence queries, membership queries that α has written down on its oracle tape, and the counterexamples received. The query complexity measures the amount of information transfer between the teacher and the learner. On the other hand, the *time complexity* of α is the number of time steps it takes before it outputs a representation of the target concept. We will focus on time complexity. $\mathcal{P}$ is said to be polynomial-time learnable, if there are a polynomial q and a learner α learning $\mathcal{P}$, such that for all n, and for all $c \in \mathcal{P}_n$, the time complexity of learner α learning the target concept c, on input $\langle 0^n, 0^l \rangle$ is bounded by $q(n + l)$. Adapting the complexity theoretic notation FP for functional polynomial time, we denote polynomial time learnability by FP-learnability.

2.2 Definitions related to the teaching assistant model

We now give formal definitions of the teaching assistant model. We start with the abstract definition of teaching assistants.

Definition 2.1 Let $\mathcal{P}$ be a representation class. Let $Q : \mathbf{N} \times \mathbf{N} \times \Sigma^* \times 2^{\Sigma^*} \to \{0, 1\}$ be a predicate. A tuple $\langle n, l, x, c \rangle \in \mathbf{N} \times \mathbf{N} \times \Sigma^* \times 2^{\Sigma^*}$, is said to be $\mathcal{P}$-valid if $c \in \mathcal{P}_n$ and $size(c) \leq l$. The *teaching assistant* defined by Q w. r. t. the representation class $\mathcal{P}$ is the set $L(\mathcal{P}) = \{\langle n, l, x, c \rangle | \langle n, l, x, c \rangle$ is P-valid and $Q(n, l, x, c)\}$.

Now we formalize learning using teaching assistants. Let $\mathcal{P} = \langle R, \mu, p \rangle$ be a representation class and $L(\mathcal{P})$ be a teaching assistant for $\mathcal{P}$. As in Angluin's model, learners are deterministic oracle Turing machine transducers. The learning algorithm comprises of a learner and teaching assistant pair. The teacher selects a target concept $c \in \mathcal{P}_n$. The learner α, on input $\langle 0^n, 0^l \rangle$, has to output a representation $r \in R$ such that $\mu_n(r) = c$, provided $size(c) \leq l$. The learner α can query its teaching assistant, say $L(\mathcal{P})$. Notice that when α wants to query the teaching assistant about $\langle n, l, x, c \rangle$, it is enough that α communicates x to the teaching assistant, since other parameters are implicit. For such a query x by the learner, the learner receives the answer 'Yes' if $\langle n, l, x, c \rangle \in L(\mathcal{P})$ and 'No' if $\langle n, l, x, c \rangle \notin L(\mathcal{P})$. $\mathcal{P}$ is said to be learnable with assistant $L(\mathcal{P})$ if there is a learner α for $\mathcal{P}$ with queries to $L(\mathcal{P})$ as teaching assistant and we say α learns $\mathcal{P}$ with teaching assistant $L(\mathcal{P})$.

Let $\mathcal{P}$ be a representation class. Let α be a learner learning $\mathcal{P}$ with a teaching assistant $L(\mathcal{P})$. For an input $\langle 0^n, 0^l \rangle$, the *time complexity* of a learner is the number of steps it takes before it writes down a representation of the target concept. $\mathcal{P}$ is said to be deterministic polynomial-time learnable (FP-learnable) with a teaching assistant $L(\mathcal{P})$, if there is a polynomial q and a deterministic learner α learning $\mathcal{P}$ with $L(\mathcal{P})$ such that for all n, and for all $c \in \mathcal{P}_n$, on input $\langle 0^n, 0^l \rangle$, the running time of learner α learning c, is bounded by $q(n + l)$.

In order to quantify the complexity of teaching assistants we can define classes of teaching assistants with respect to a given representation class. These classes can be defined analogous to some well-studied complexity classes. In this paper we are interested in the teaching assistant class analogous to the complexity class SPP which we define next.

Definition 2.2 Let $\mathcal{P}$ be a representation class and $L(\mathcal{P})$ be a teaching assistant. $L(\mathcal{P})$ is said to be in the class SPP($\mathcal{P}$) if there exists a polynomial-time nondeterministic oracle

Turing machine M which for any $\mathcal{P}$-valid tuple $\langle n, l, x, c\rangle$, on input $\langle 0^n, 0^l, x\rangle$ uses c as oracle and produces a gap=1 if $\langle n, l, x, c\rangle \in L(\mathcal{P})$ and gap=0 if $\langle n, l, x, c\rangle \notin L(\mathcal{P})$. The behavior of machine M is not specified on inputs that are not $\mathcal{P}$-valid.

Finally we have the following definition of learnability with teaching assistant classes.

Definition 2.3 Let $\mathcal{C}$ denote a teaching assistant class. We say a representation class $\mathcal{P}$ is FP-learnable with a $\mathcal{C}$-assistant if there exists a teaching assistant $L(\mathcal{P}) \in \mathcal{C}(\mathcal{P})$ and an FP-learner α, such that α learns $\mathcal{P}$ with the teaching assistant $L(\mathcal{P})$.

The following theorem relates the TA model to Angluin's model.

Theorem 2.4 ([AV00]) *For any representation class $\mathcal{P}$*

1. *$\mathcal{P}$ is* FP*-learnable with membership queries if and only if it is* FP*-learnable with a* P*-assistant.*
2. *If $\mathcal{P}$ is* FP*-learnable with membership and equivalence queries then it is* FP*-learnable with a* NP*-assistant.*

3 Learning using SPP assistants

In this section, we give learning algorithms using teaching assistants in the class SPP. First we consider the problem of learning permutation groups. The representation class SYM of permutation groups is defined formally below.

The *symmetric group* S_n consists of all permutations on {1,. . . ,n} with permutation composition as group operation. Subgroups of S_n are called *permutation groups*. We can encode permutations in S_n with binary strings of length $4n\lceil \log n\rceil$. Let $\textsc{Sym} = \langle R, \{\mu_n\}_{n\geq 1}, t\rangle$ denote the representation class of permutation groups where $R = \cup_{n\geq 1} R_n$, and $r \in R_n$ encodes a set of permutations of S_n, namely, it is a concatenation of some strings, each encoding a permutation as explained above. For $r \in R_n$, $\mu_n(r)$ is the subgroup of S_n generated by permutations encoded in r. Here, $t(n) = 4n\lceil \log n\rceil$.

The SPP learning algorithm for permutation groups very closely follows the SPP algorithm for Graph Isomorphism problem given by Arvind and Kurur [AK02]. In [AK02], the upper bound for Graph Isomorphism obtained by placing a more general problem of FIND GROUP (where the problem is to construct a generator set for an unknown permutation group using polynomial time membership test) in the oracle function class FP^{SPP}. We observe that this algorithm can be adapted in a straightforward manner to obtain a learning algorithm which uses an SPP assistant. We just state the result in this extended abstract, omitting the details.

Theorem 3.1 *The representation class* SYM *is* FP*-learnable with an* SPP*-assistant.*

Monotone DNF

Notice that from Theorem 2.4 and the fact that monotone DNFs are learnable with membership and equivalence queries, they are learnable with an NP-assistant. Now we present a learning algorithm which uses an SPP-assistant.

Let mDNF denotes the concept class of monotone functions represented by monotone DNF formulae. For easiness of presentation, we define the size of a concept c in mDNF as the number of prime implicants of c. This is only polynomially different from the definition given earlier.

Theorem 3.2 *The representation class* mDNF *of monotone* DNF *formulae is* FP-*learnable with an* SPP-*assistant.*

Proof. (*Sketch*) We will first present a teaching assistant $L(\text{mDNF})$ and show that there exists an oracle machine M with the required gap behavior accepting $L(\text{mDNF})$. Then we present a learner Learner-mDNF which uses $L(\text{mDNF})$ to output a monotone formula for the target concept.

The learning algorithm that we give is an adaptation of Angluin's algorithm for learning monotone DNF using membership and equivalence queries. The algorithm works as follows. Let c be the concept represented by a monotone formula ϕ. As in the case of Angluin's algorithm for mDNF, the learning algorithm will find out each of the prime implicants of ϕ one by one. This is done by doing a prefix search with the aid of the teaching assistant. Note that if we just need an NP-assistant, it is rather straightforward to define one (proof of Theorem 2.4) for this purpose. In order to make the TA in SPP, the prime implicants are prefix searched in the order of increasing number of variables in them. The teaching assistant that we use for the learner is defined as:

$L(\text{mDNF}) = \{\langle n, l, \phi, k, y, c\rangle | \phi$ is a disjunction of monotone terms; there are no $x \in (c \setminus \phi)$ having less than k 1s; there exists $z \in (c \setminus \phi)$ having exactly k 1s; y is a prefix of lex-first such $z\}$

Now we will show that $L(\text{mDNF}) \in \text{SPP}(\text{mDNF})$.

Claim 3.2.1 $L(\text{mDNF}) \in \text{SPP}(\text{mDNF})$

Proof. [of claim] The oracle nondeterministic machine M that witnesses $L(\text{mDNF})$ in $\text{SPP}(\text{mDNF})$ is defined below. M uses an auxiliary oracle nondeterministic machine N_2 as subroutine which in turn uses N_1. We define these two auxiliary machines first.

Machine$N_1^c(\langle n, l, \phi, i, k, y\rangle)$
1. **Guess** in lex-first order strings $z_1, z_2, \ldots, z_i$
2. **Accept** if all the following conditions are met
3. $\quad y$ is a prefix of z_1
4. $\quad z_1, z_2, \ldots, z_i$ have exactly k 1s
5. $\quad z_1, z_2, \ldots, z_i$ are in c (using membership queries to c)
6. $\quad z_1, z_2, \ldots, z_i$ are not in ϕ
7. **Else Reject**

Machine$N_2^c(\langle n, l, \phi, k, y\rangle)$
1. Produce a gap of $\sum_{i=1}^{l} acc_{N_1^c}(\langle n, l, \phi, i, k, y\rangle) \prod_{j=i+1}^{l} \left(1 - acc_{N_1^c}(\langle n, l, \phi, j, k, y\rangle)\right)$

Machine $M^c(\langle n, l, \phi, k, y\rangle)$
1. If ϕ is not a disjunction of monotone terms produce a gap=0
2. Produce a gap of $gap_{N_2^c}(\langle n, l, \phi, k, y\rangle) \prod_{k'=1}^{k-1} \left(1 - gap_{N_2^c}(\langle n, l, \phi, k', y\rangle)\right)$

Let c be a target concept fixed by the teacher with $size(l) \leq l$. Let k_0 be the number of variables in the smallest (in terms of the number of variables) prime implicant of c that is not in ϕ. Let i_0 be the number of such prime implicants with k_0 variables.

We will argue that M produces a gap=1 if $(\langle n, l, \phi, k, y, c\rangle) \in L(\text{mDNF})$ and produces a gap=0 if $(\langle n, l, \phi, k, y, c\rangle) \notin L(\text{mDNF})$. Since if ϕ not a disjunction of monotone terms M produces a gap=0, we will assume otherwise for the rest of the proof. $\langle n, l, \phi, k, y, c\rangle \in L(\text{mDNF})$ if and only if $k = k_0$ and y is a prefix of lex-first string $z \in (c \setminus \phi)$ with exactly k ones. We need to show that M^c produces a gap=1 in this case and produce a gap=0 otherwise. For easiness of presentation, we only present the case when y is the prefix of lex-first string $z \in (c \setminus \phi)$ with exactly k ones. The other case can be argued similarly.

We first observe certain acceptance behavior of N_2 which is stated as the next claim. The proof follows from the description of N_1 and N_2.

Claim 3.2.2 (subclaim)

1. *If* $k < k_0$, $gap_{N_2^c}(\langle n, l, \phi, k, y\rangle) = 0$.
2. *If* $k = k_0$, $gap_{N_2^c}(\langle n, l, \phi, k, y\rangle) = 1$.

Proof. [sketch for subclaim]

1 $k < k_0$. In this case N_1^c will have 0 accepting paths. Hence whatever be the value of i, $gap_{N_1^c}(\langle n, l, \phi, i, k, y\rangle) = 0$. Hence the total gap produced by $N_2^c = 0$.

2 $k = k_0$. Split the sum in line 1 of description of N_2^c into three: $i < i_0$, $i = i_0$, and $i > i_0$. For $i > i_0$, N_1^c will have 0 accepting paths and hence that part of the sum contributes 0 to the overall gap. For $i = i_0$, N_1^c will exactly 1 accepting path and for this case each of terms in the product contributes 1. Thus the contribution to the overall gap in this case is 1. For the case $i < i_0$, consider the term $\prod_{j=i+1}^{l} \left(1 - acc_{N_1^c}(\langle n, l, \phi, j, k, y\rangle)\right)$. Since $i < i_0$, there is term in this product with $j = i_0$. For this value of $j = i_0$, $acc_{N_1^c}(\langle n, l, \phi, i_0, k, y\rangle) = 1$ and hence $\left(1 - acc_{N_1^c}(\langle n, l, \phi, i_0, k, y\rangle)\right) = 0$. Hence the contribution from this case to the overall gap =0. There for the total gap=1.

■

Case 1: $k = k_0$. In this case we need to show that $gap_{M^c} = 1$. Consider the gap expression in the line 2 of the description of M^c. From the above claim, $gap_{N_2^c}(\langle n, l, \phi, j, k, y\rangle) = 1$ and for all $k' \leq k - 1$, $gap_{N_2^c}(\langle n, l, \phi, j, k', y\rangle) = 0$. Hence the overall gap produced is 1.
Case 2: $k < k_0$. We need to show that $gap_{M^c} = 0$. From the above claim, the term $gap_{N_2^c}(\langle n, l, \phi, j, k, y\rangle) = 0$ and hence the overall gap produced is 0.
Case 3: $k > k_0$. (This case is similar to the case $i < i_0$ in the above claim) In this case we need to show that $gap_{M^c} = 0$. Consider the product term $\prod_{k'=1}^{k-1} \left(1 - gap_{N_2^c}(\langle n, l, \phi, k', y\rangle)\right)$ of the gap expression in the line 2 of the description of M^c. Since k' runs from 1 to $k - 1$, if $k > k_0$ there is a term $\left(1 - gap_{N_2^c}(\langle n, l, \phi, k_0, y\rangle)\right)$ that appear in this product. Now from the subclaim, for this value of k', $\left(1 - gap_{N_2^c}(\langle n, l, \phi, k_0, y\rangle)\right) = 0$ and the product term, and consequently the overall gap, is 0.

■

The following algorithm learns mDNF in polynomial time by making only $O(ln)$ queries to the teaching assistant L(mDNF). In the algorithm, for a binary string $y \in \{0,1\}^n$ Y denotes the the corresponding monotone term.

LEARNER-mDNF$(0^n, 0^l)$
1. $\phi \leftarrow \mathbf{F}$
2. $k \leftarrow 1$
3. **while** $\langle 0^n, \phi, k, \lambda, c\rangle \notin L$
4. $\quad k \leftarrow k+1$
5. **if** $k = n+1$
6. $\quad$ **then** output ϕ and stop
7. $y \leftarrow \lambda$
8. **for** $j = 1$ to n **do**
9. $\quad$ **if** $\langle 0^n, \phi, k, y1, c\rangle \in L$
10. $\quad\quad$ **then** $y \leftarrow y1$
11. $\quad\quad$ **else** $y \leftarrow y0$
12. $\phi \leftarrow \phi \vee Y$
13. **goto** 2

■

4 Conclusions and open problems

This paper proves new upper bounds on learning two concept classes; permutation groups and monotone DNFs, in the teaching assistant model of learning: both these classes are shown to be learnable with SPP assistants. While the upper bound for permutation groups solves an open problem stated in [AV00], the upper bound for monotone DNF illustrates that teaching assistant model can be successfully used to study the complexity of learning important and well-studied concept classes. There are many open issues related to the teaching assistant model. Are there other concept classes that are learnable with SPP assistants? Perhaps the most important concept class that is worth investigating in the TA model is general DNF formulae. The major open problem regarding DNF learning in Angluin's model is whether equivalence + membership queries are powerful enough for learning DNF formulae. A natural step towards progress in this problem is to investigate it in a more general learning framework than Angluin's model. The TA model with NP teaching assistants provides exactly such a framework. Are general DNF formulae learnable using an NP assistant?

Acknowledgements.

I would like to thank John Hitchcock for helpful discussions.

References

[AK02] V. Arvind and P. Kurur. Graph isomorphism is in SPP. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 743 –750, 2002.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[Ang90] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[AV96] V. Arvind and N. V. Vinodchandran. The complexity of exactly learning algebraic concepts. In *Proceedings of the 7th International Workshop on Algorithmic Learning Theory*, volume 1160 of *Lecture Notes in Artificial Intelligence*, pages 100–112, 1996.

[AV00] V. Arvind and N. V. Vinodchandran. Exact learning via teaching assistants. *Theoretical Computer Science*, 241:51–81, 2000. Special issue devoted to the 7th International Workshop on Algorithmic Learning Theory.

[BCG00] J. Balcázar, J. Castro, and D. Guijarro. Abstract combinatorial characterizations of exact learning via queries. In *Proceedings of Thirteenth Annual ACM Conference on Computatioanl Learning Theory*, pages 248–254, 2000.

[BCG01] J. Balcázar, J. Castro, and D. Guijarro. A general dimension for exact learning. In *Proceedings of the Fourteenth Annual ACM Conference on Computational Learning Theory*, volume 2111 of *Lecture Notes in Artificial Intelligence*, pages 354–367, 2001.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity – I & II*. Springer Verlag, Berlin Hiedelberg, 1988.

[Bei94] R. Beigel. Perceptrons, PP, and the polynomial hierarchy. *Computational Complexity*, 4(4):339–349, 1994.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[Gav93] R. Gavaldà. On the power of equivalence queries. In *Proceedings of the First European Conference on Computational Learning Theory*, pages 193–203, 1993.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key crypto-systems. *SIAM Journal on Computing*, 17(2):309–355, 1988.

[Gup95] S. Gupta. Closure properties and witness reduction. *Journal of Computer and System Sciences*, 50(3):412–432, 1995.

[Heg95] H. Hegedüs. Generalized teaching dimensions and the query complexity of learning. In *Proceedings of the Eighth ACM Conference on Computational Learning Theory*, pages 108–117, 1995.

[Hit03] J. M. Hitchcock. The size of SPP. 2003. Manuscript submitted.

[HPRW96] L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan, and D. Wilkins. How many queries are needed to learn? *Journal of the Association for Computing Machinery*, 43(5):840–862, 1996.

[KST92] J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Journal of Computational Complexity*, 2:301–330, 1992.

[KvM02] A. Klivans and D. van Melkebeek. Graph nonisomorhism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31:1501–1526, 2002.

[OH93] M. Ogiwara and L. Hemachandra. A complexity theory of feasible closure properties. *Journal of Computer and System Sciences*, pages 295–325, 1993.

[Val76] L. Valiant. Relative complexity of checking and evaluating. *Information Processing Letters*, 5:20–23, 1976.

[Vin97] N. V. Vinodchandran. Improved lowness results for solvable black-box group problems. In *Proceedings of the Seventeenth International Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1346 of *Lecture Notes in Computer Science*, pages 220–234, 1997.

[Vin99] N. V. Vinodchandran. *Counting Complexity and Computational Group Theory*. PhD thesis, University of Madras, Chennai, India, 1999. Also published at *PhD Theses series of ECCC*, URL: `http://www.eccc.uni-trier.de/eccc-local/ECCC-Theses`.